2016 年度 修士論文

グラフ書換え言語 LMNtal における グラフ型検査およびルール型保存性検査

提出日： 2017 年 1 月 25 日
指導： 上田 和紀 教授

早稲田大学 基幹理工学研究科
情報理工・情報通信専攻

学籍番号：5115F066-1
吉元 佑介

概要

計算機ネットワークや，ソーシャルネットワーク上の人間関係などは，グラフ構造として自然にモデル化することができる．グラフを基本的なデータ構造とする言語モデルは，これらの対象を自然に扱うことができるため重要である．そのような言語モデルの一つに，LMNtal がある．LMNtal は，数学的な簡潔さと高い表現力を兼ね備えた言語モデルであるが，プログラミング言語としての LMNtal に，静的な型検査機能は存在しない．静的な型検査ができれば，プログラムの良い性質を保証できる他，型情報を利用した実行時性能の向上といった応用が考えられる．

そこで，LMNtal の型体系として，グラフ書き換え系の型体系である Shape Type を導入した．Shape Type によって，グラフが特定の構造を持っているかどうかの検査（グラフの型検査），また，グラフの書き換え規則がその構造を崩さないかどうかの検査（ルールの型検査）を行える．LMNtal 上での検査を行うために，まず，Shape Type を，LMNtal に導入できる形で定義した．そして，特にグラフ型検査に関しては，アルゴリズムを具体的に提示し，その停止性と健全性の証明を与えた．また，ルール型検査に関しては，アルゴリズムの概略を示した．

Abstract

We can model computer networks or relationships among people on social networks as graphs. Language models which use graphs as the primitive data structure are important, because they can handle such models naturally. LMNtal is one of such languages. LMNtal enjoys mathematical simplicity and high expressive power, but LMNtal as a programming language does not feature static type checking. Static type checking would assure good invariants of programs and allows the use of type information to speed up execution.

To achieve these advantages, we introduced Shape Types, a type system for graph rewriting systems, to LMNtal. Shape Types allow us to check whether a graph has a specific structure, and whether a graph rewriting rule preserves the structure. For the type checking on LMNtal, we defined a type system based on Shape Types for LMNtal. We showed an algorithm for type checking of graphs, and gave a proof of its termination and soundness. We also showed an outline of algorithm for type checking of rules.

目次

図目次

第 1 章

はじめに

1.1 研究の背景と目的

計算機ネットワークや，ソーシャルネットワーク上の人間関係などは，グラフ構造として自然にモデル化することができる．グラフを基本的な要素とする言語モデルは，これらの対象を自然に扱うことができるため重要である．そのような言語モデルの一つに，LMNtal [1] がある．LMNtal では，プログラム全体が，グラフと，グラフを書き換えるためのルールの集合によって表現される．LMNtal は，言語モデルとして十分に単純であるため，数学的な扱いが容易である．それでいて，グラフではなく式を基本的な要素とする既存の言語モデルを，うまくエンコードできる程度の表現力を有している．また，LMNtal のグラフ書き換えを実行する処理系が存在しており，計算機上で実行されるプログラミング言語として LMNtal を用いることもできる．

しかしながら，プログラミング言語として LMNtal には，静的な型検査機能が存在しない．型検査を行うことができれば，プログラムの良い性質を保証することができる．例えば，LMNtal で，グラフによってエンコードされたリストの，先頭に要素を追加するルールを書いたとする．リストにそのルールを適用したとき，結果として得られるグラフはリストとなっていなければならないが，型検査によって，そのことをコンパイラがチェックできる．また，LMNtal プログラム実行時に，型情報を利用した高速化を行うことも可能であろう．

そこで，本研究では，LMNtal に静的型を導入し，計算，すなわちグラフの書き換えが，ある種の条件を満たすことを保証できるようにする．既存の言語，特に，型体系の発達した関数型言語とは異なり，LMNtal におけるプログラムは式ではなく，グラフと書き換え規則の集合である．したがって，関数型言語の型体系を模倣して型を導入することは難し

い．そこで，グラフの型として考案された，Shape Type [2] を用いて，静的型検査を行うことを考える．

Shape Type を用いることで，グラフが特定の構造を持っているかどうかを検査できるようになる．この検査をグラフの型検査と称する．また，ルールがグラフの構造を崩さないかどうかも検査できる．これを，ルールの型検査と称する．本研究では，これらの検査に焦点を当て，アルゴリズムの定式化を試みた．グラフの型検査に関しては，アルゴリズムの定式化を行い，また，アルゴリズムの停止性と健全性を証明した．

1.2 本論文の構成

第 2 章では，Shape Type を導入する対象言語である LMNtal を定義する．第 3 章では，LMNtal のための Shape Type を定義し，グラフとルールの型検査のアルゴリズムを説明する．第 4 章では本研究をまとめ，判明した問題点や今後の展開を述べる．

第 2 章

LMNtal

型付けの対象とする言語である LMNtal を定義する．これは，LMNtal のサブセットである Flat LMNtal [1] に，ルールの移動を許さないという制限を加えたものに相当する．

2.1 構文

LMNtal の構文は，アトム，リンク，グラフ，ルールセット，プロセスの，5 つの構文要素から構成される．

2.1.1 アトム・リンク

アトムとは，グラフ構造のノードを表すものである．アトムの名前は，小文字から始まる識別子で表される．リンクとは，グラフ構造の無向エッジを表すものである．リンクの名前は，大文字から始まる識別子で表される．ひとつのアトムに何本のリンクが接続できるかは予め定まっているため，アトム名と，接続できるリンク本数との組を指定することによって，ひとつの種類のアトムが指定される．この，アトム名と接続可能リンク数の組を，ファンクタと呼ぶ．ファンクタ (p, n) を持つアトムは，「n 価の p アトム」と称する．また，誤解がないときには，単に「p アトム」と称する．ファンクタとして用いることができるものの集合を，Functor $=$ AtomNames $\times \mathbb{N}$ で表す．ただし，AtomNames は適当な可算無限集合，$\mathbb{N}$ は 0 を含む自然数全体の集合である．

2.1.2 グラフ

グラフの構文は，以下のように定義される．

$$
\begin{array}{rcll}
X & ::= & 0 & \text{(空)} \\
 & | & p(L_1, ..., L_n) & \\
 & & \text{ただし，} (p, n) \in \text{Functor} & \text{(アトム)} \\
 & | & X, X & \text{(分子)}
\end{array}
$$

空は，何もないグラフを表す．アトム $p(L_1, ..., L_n)$ は，ファンクタ (p, n) を持つアトムに，n 本のリンク $L_1, ..., L_n$ が接続されていることを表す．ただし，L_1 から L_n までのリンクは順序付けられており，リンクの順番の異なるアトムは異なるグラフとみなされる．なお，「異なる」とは，後に定義する合同関係の上で，「合同でない」という意味である．分子は，2 つのグラフを合併したグラフを表す．例えば，グラフ alice(L), bob(L) は，1 価のアトム alice と bob が，リンク L によって接続されているグラフを表す．これを，図 2.1 のように示す．

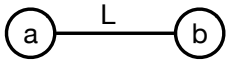

図 2.1　簡単な LMNtal グラフの例

グラフを図示する際，必要がないときは，アトム名の 2 文字目以降を省略することとする．

なお，この構文規則で生成される全てのものがグラフとして認められるわけではない．以下の「グラフに関するリンク条件」を満たすものだけがグラフである．

> グラフに関するリンク条件：
> グラフには，同名のリンクが 2 回を超えて出現してはならない

グラフ中に 1 回しか出現しないリンクを自由リンクという．これは，片端はアトムに接続されているが，もう片端はアトムに接続されておらず宙に浮いているようなリンクを表す．グラフ中に 2 回出現するリンクを局所リンクという．これは，両端がアトムに接続されているリンクを表す．リンク条件は，一つのリンクが 3 つ以上のアトムを接続してはならないということを意味する．

自由リンクを持たないグラフを，閉じたグラフと呼ぶ．

グラフの例

関数型言語で使われるリストを，LMNtal グラフで表現することができる．以下のグラフを考える．

$$\texttt{cons}(\texttt{I1},\texttt{L1},\texttt{LIST}),\texttt{int}(\texttt{I1}),\texttt{cons}(\texttt{I2},\texttt{L2},\texttt{L1}),\texttt{int}(\texttt{I2}),\texttt{nil}(\texttt{L2})$$

リストの終端は 1 価の `nil` アトムで，cons セルは 3 価の `cons` アトムで，リストに付随するデータは 1 価の `int` アトムで表現されている．グラフは自由リンク `LINK` を持ち，`LINK` につながっているグラフ全体がリスト構造となっている．このグラフを図示すると，図 2.2 のようになる．

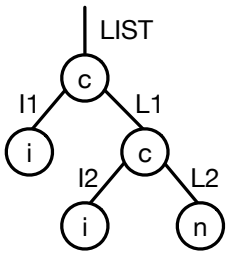

図 2.2　LMNtal グラフの例

グラフ全体の集合を Graph で表す．また，$F \subseteq$ Functor なる F に属するものだけを，アトムの出現として許したグラフ全体を，Graph(F) で表す．ただし，F は，のちに述べる=アトムのために，$(=,2)$ を含んでいる必要がある．

2.1.3　ルールセット

ルールセットの構文は，以下のように定義される．

$$\begin{array}{rcll} R & ::= & 0 & (\text{空}) \\ & | & X \text{ :- } X & (\text{ルール}) \\ & | & R, R & (\text{分子}) \end{array}$$

ルールは，「:- 記号の左辺に現れるグラフを，右辺に現れるグラフに書き換える」という書き換え規則を表す．

ただし，以下の「ルールに関する回数条件」を満たすものだけがルールとみなされる．

ルールに関する回数条件：
ルールに出現するリンクは，ちょうど 2 回出現しなければならない．

これは，書き換えによって，自由リンクの生成・消滅が起こらないことを意味する．

2.1.4 プロセス

プロセスの構文は，以下のように定義される．

$$P ::= X, R$$

つまり，プロセスはグラフとルールセットの組である．

2.1.5 表記上の注意

表記上の簡便をはかり，また，文字列表現が指すプロセスを一意に定めるために，以下の規則を用いる．

- 0 価のアトムの括弧は省略できる．
- グラフやルールセットに出現するカンマは左結合とする．
- グラフやルールセットやプロセスに出現するカンマは，:- 記号より強く結合するものとする．
- 結合順を変更または明示するために括弧を用いる．

2.2 意味論

2.1 節では，グラフやルールの意味するところを直感的に述べてきたが，ここで厳密な意味論を与える．まず，グラフやルールセットが「同じものである」ということを，グラフ・ルールセットの合同規則として定義する．

グラフの合同関係 $\equiv$ は，図 2.3 を満たす最小の同値関係である．

(Equiv1) は，グラフと空グラフで分子を作ったものは，元のグラフと合同であることを意味する．

(Equiv2), (Equiv3) は，分子の交換律と結合律を表す．これらの規則によって，複数のグラフ X_1，X_2，…，X_n から分子を作る際，合同なものの違いを無視すれば，できるグラフは一意に定まる．

$$0, X \equiv X \tag{Equiv1}$$

$$X_1, X_2 \equiv X_2, X_1 \tag{Equiv2}$$

$$X_1, (X_2, X_3) \equiv (X_1, X_2), X_3 \tag{Equiv3}$$

$$X \equiv X[L/M]$$
ただし，L は X の局所リンク (Equiv4)

$$\frac{X_1 \equiv X_1'}{X_1, X_2 \equiv X_1', X_2} \tag{Equiv5}$$

$$=(L, L) \equiv 0 \tag{Equiv6}$$

$$=(L_1, L_2) \equiv =(L_2, L_1) \tag{Equiv7}$$

$$=(L, L'), p(L_1, ..., L, ..., L_n) \equiv p(L_1, ..., L', ..., L_n)$$
ただし，L は $p(L_1, ..., L, ..., L_n)$ の自由リンク (Equiv8)

図 2.3　LMNtal グラフの合同関係

(Equiv4) は，局所リンク名のみが異なるグラフは同じグラフであることを意味する．これは，関数型言語での，束縛変数の α 変換に相当する．なお，$X[L/M]$ とは，X に現れる全ての L の出現を M に置換したグラフである．このことを利用して，以下の略記法を定義する．

グラフ

$$p_1(L_{11},...,p_2(L_{21},...,L_{2m}),...,L_{1n})$$

は，使用されていない適当なリンク名 L を用いて，

$$p_1(L_{11},...,L,...,L_{1n}),p_2(L_{21},...,L_{2m},L)$$

と表せるグラフの一つである

また，グラフを図示する際，必要がないときは，局所リンク名の記載を省略する．

なお，自由リンクに関しては，リンク名を変更したグラフは合同とはみなされない．これは，関数型言語において，自由変数の名前を変更した式は，元の式とは異なるものとみなされることに相当する．

(Equiv5) は，2 つの合同なグラフに，同一のグラフを追加して分子を作っても，できた 2 つのグラフは合同であることを意味する．

(Equiv6) 〜 (Equiv8) は，予約ファンクタ $(=,2)$ の性質を表している．なお，=は小文字から始まる識別子ではないが，特別にアトム名として認める．2 価の =アトムに接続されている 2 本のリンクは，=アトムを介さずに直接接続されているとみなされる．(Equiv6) は，環状になった 1 本のリンクは，他のアトムを接続することはないため，空とみなせることを意味する．(Equiv7) は，2 価の =アトムの性質上，リンク順は無視できることを意味する．(Equiv8) は， =アトムによってリンクが直接接続されていることを規定している．例えば，$\mathtt{a(L1)},\mathtt{=(L1,L2)},\mathtt{b(L2)}$ は，$\mathtt{a(L2)},\mathtt{b(L2)}$ と合同である．すなわち，リンク L1 とリンク L2 は直接つながっているので，同じリンクであるとみなせるのである．

同様に，ルールの同値関係 $\equiv$ は，図 2.4 を満たす最小の同値関係である．

プロセスの同値関係は，以下をもって定義される．

$$P_1 = X_1, R_1 \quad P_2 = X_2, R_2 \quad \text{とすると}$$
$$P_1 \equiv P_2 \ \Leftrightarrow \ X_1 \equiv X_2 \text{ かつ } R_1 \equiv R_2$$

プロセスはグラフとルールセットからなり，グラフ部分をルールを用いて書き換えることができる．どのような書き換えが許されるのかを，プロセス間の遷移規則として定義する．プロセスの遷移関係 $\rightarrow$ は，図 2.5 を満たす最小の二項関係である．

(Trans1) は，プロセス X_1, R_1 が，X_1', R_1' に遷移することが分かっているとき，グラフやルールセットを新たに加えても，遷移するという関係は保たれることを意味する．

(Trans2) は，合同関係によってプロセスを変形しても，遷移関係は保たれることを意味

$$0, R \equiv R \tag{Equiv1}$$

$$R_1, R_2 \equiv R_2, R_1 \tag{Equiv2}$$

$$R_1, (R_2, R_3) \equiv (R_1, R_2), R_3 \tag{Equiv3}$$

$$R \equiv R[L/M] \tag{Equiv4}$$

$$\frac{R_1 \equiv R_1'}{R_1, R_2 \equiv R_1', R_2} \tag{Equiv5}$$

図 2.4　LMNtal ルールセットの合同関係

$$\frac{X_1, R_1 \to X_1', R_1'}{X_1, X_2, R_1, R_2 \to X_1', X_2, R_1', R_2} \tag{Trans1}$$

$$\frac{P_1' \equiv R_1 \quad P_1' \to P_2' \quad P_2' \equiv P_2}{P_1 \to P_2} \tag{Trans2}$$

$$X_1, (X_1 \text{ :- } X_2) \to X_2, (X_1 \text{ :- } X_2) \tag{Trans3}$$

図 2.5　LMNtal プロセスの遷移関係

する．

(Trans3) は，X_1 というグラフと，「X_1 を X_2 に書き換える」というルールからなるルールセットを組にしたプロセスは，X_2 に遷移することを意味する．この規則が，遷移の実質的な意味を定めている．

$X, R \to X', R'$ が成立することを，「X を R によって X' に書き換える」，「X に R を

適用して X' を得る」と表現する．

2.2.1 プロセスの遷移例

グラフ

$$\texttt{list}(\texttt{LIST}),\ \texttt{cons}(\texttt{int},\texttt{nil},\texttt{LIST}),$$

を考える．これは，1 価アトム list によって，先頭を明示したリストである．このグラフを X とおく．また，ルール

$$\texttt{list}(\texttt{LINK})\ \texttt{:-}\ \texttt{list}(\texttt{cons}(\texttt{int},\texttt{LINK}))$$

を考える．これは，リストの先頭に，データを 1 つ追加するルールである．このルールのみからなるルールセットを R とおく．

X に R を適用すると，

$$\texttt{list}(\texttt{LIST}),\ \texttt{cons}(\texttt{int},\texttt{cons}(\texttt{int},\texttt{cons}(\texttt{int},\texttt{nil}),\texttt{LIST})$$

が得られることを示す（このグラフを X' とおく）．(Equiv4) より，

$$R \equiv \texttt{list}(\texttt{LIST})\ \texttt{:-}\ \texttt{list}(\texttt{cons}(\texttt{int},\texttt{LIST})) \tag{1}$$

である（右辺を R' とおく）．(Trans3) より，

$$\texttt{list}(\texttt{LIST}), R' \rightarrow \texttt{list}(\texttt{cons}(\texttt{int},\texttt{LIST})), R' \tag{2}$$

である．(2) と (Trans1) より，

$$X, R' \rightarrow X', R' \tag{3}$$

である．(1)，(3) と (Trans3) より，

$$X, R \rightarrow X', R$$

を得る．

第 3 章

LMNtal Shape Type

LMNtal Shape Type は，Shape Type [2] を LMNtal の構文と意味に即して変形したものである．LMNtal Shape Type の基本的な考え方は，グラフの集合を生成文法で表現することである．つまり，あるグラフを開始記号として定め，そのグラフを，生成規則であるルールセットによって書き換えることで，集合の要素となるグラフを生成してゆく．ただし，文法が文脈自由文法とみなせるよう，開始記号や生成規則の形に制限を加える．

3.1 構文

LMNtal Shape Type G の構文を，以下のように定義する．

$$\begin{aligned} G &::= s(LS_1, ..., LS_n).Prod. \\ Prod &::= p(L_1, ..., L_n) \Rightarrow X \\ &\quad | \quad Prod.Prod \end{aligned}$$

ただし，G について，$s(LS_1, ..., LS_n)$ はアトムであり，グラフに関するリンク条件を満たすとする．また $(s, n) \neq (=, 2)$ とする．さらに，$Prod$ について，$p(L_1, ..., L_n)$:- X はルールとなり，p および X に 2 価の =アトムが含まれてはならない．

グラフ $s(LS_1, ..., LS_n)$ は開始記号を表す．$Prod$ は生成規則の集合を表す．

3.2 意味論

LMNtal Shape Type G 型のグラフに出現できるファンクタは，$Prod$ 中の生成規則の，右辺に現れ，左辺には現れないファンクタ，及び，$(=, 2)$ である．これは，文字列の生成文法で言う，終端記号に相当する．このようなファンクタ全体の集合を，$\mathrm{Term}(G)$ で表す．

また，生成規則集合 Prod をルールセットとみなせるよう，変換関数 Induction を，以下のように定義する．

$$\begin{aligned} \mathrm{Induction}(p(L_1,...,L_n) \Rightarrow X) &= p(L_1,...,L_n) \text{ :- } X \\ \mathrm{Induction}(Prod_1.Prod_2) &= \mathrm{Induction}(Prod_1), \mathrm{Induction}(Prod_2) \end{aligned}$$

そして，グラフと LMNtal Shape Type の型付け関係 : を定義する．まず，補助的な型付け関係 $\lhd$ を定義する．

$$s(LS1,...,LSn) \lhd G(\mathrm{Typing1}) \qquad (\mathrm{Typing1})$$

$$\frac{X' \lhd G \quad X', \mathrm{Induction}(Prod) \to X, \mathrm{Induction}(Prod)}{X \lhd G} \qquad (\mathrm{Typing2})$$

型付け関係 : は，以下のように定義する．

$$X \lhd G \text{ かつ } X \in \mathrm{Graph}(\mathrm{Term}(G)) \;\Leftrightarrow\; X : G$$

$X \lhd G$ なるグラフ X は，開始記号を，生成規則集合を用いて，0 回以上遷移させてできるグラフを表している．そして，そのようなグラフ X のうち，G の終端ファンクタのみから構成されるグラフが，$X : G$ という関係を満たす．

グラフ X が G 型であるとは，$X : G$ のことである．ルール *Rule* が LMNtal Shape Type G に関して型安全であるとは，G 型の任意のグラフ X について，X を *Rule* によって（可能なら）書き換えたグラフが G 型となることである．ここで，複数通りの書き換えが可能なら，全ての書き換え方について，書き換え後のグラフが G 型となることである．

なお，開始記号が自由リンク $LS_1,...,LS_n$ をもつ LMNtal Shape Type は，適当な名前 `type` を用いて，$\texttt{type}(LS_1,...,LS_n)$ と書き表すこととする．また，自由リンクを省略して，`type` と書き表すこともある．

3.2.1 例

赤黒木 [3] を LMNtal Shape Type で表現する．赤黒木は，以下の性質を持つ，ノード（内部節点，及び葉）が黒か赤に色付けされた二分探索木である．

1. 根と葉の色は黒である

2. 赤のノードは黒の子ノードを持つ
3. 任意のノードについて，そのノードから葉までの任意の路に含まれる黒のノードの数は一定である

ここで，1 と 2 を満たすような二分木を，LMNtal Shape Type `rbtree(RBTREE)` として表現する（第 4 章で，3 も表現する方法を考察する）．なお，このようなデータ構造を，一般的な関数型言語の型で表現することは難しい．その点で，LMNtal Shape Type は高い表現力を持っているといえる．

```
bnode(RBTREE).
bnode(PARENT)  ⇒  leaf(PARENT).                    (PR1)
bnode(PARENT)  ⇒  bnode(int, node, node, PARENT).  (PR2)
 node(PARENT)  ⇒  leaf(PARENT).                    (PR3)
 node(PARENT)  ⇒  bnode(int, node, node, PARENT).  (PR4)
 node(PARENT)  ⇒  rnode(int, bnode, bnode, PARENT). (PR5)
```

文法で使われているアトムの意味合いは以下の通りである．

非終端アトム：

1 価の `bnode`	黒であることが確定したノード
1 価の `node`	色が決まっていないノード

終端アトム：

1 価の `leaf`	葉
4 価の `bnode`	黒の内部節点
4 価の `rnode`	赤の内部節点
1 価の `int`	データ

開始記号を 1 価の bnode とすることで，根の色が黒であることを表している．(PR1) と (PR2) によって，黒であることが確定したノードは，葉か，黒の内部節点になれることを表している．(PR3)〜(PR5) によって，色が確定していないノードは，葉か，どちらかの色の内部節点になれることを表している．また，(PR5) によって，赤のノードの子は黒のノードでなければならないことを表している．

rbtree(RBTREE) 型のグラフの例

$$\texttt{leaf}(\texttt{RBTREE}) \quad (1)$$
$$\texttt{bnode}(\texttt{int}, \texttt{leaf}, \texttt{leaf}, \texttt{RBTREE}) \quad (2)$$
$$\texttt{bnode}(\texttt{int}, \texttt{bnode}(\texttt{leaf}, \texttt{leaf}), \texttt{leaf}, \texttt{RBTREE}) \quad (3)$$
$$\texttt{bnode}(\texttt{int}, \texttt{rnode}(\texttt{leaf}, \texttt{leaf}), \texttt{leaf}, \texttt{RBTREE}) \quad (4)$$

これらのグラフを図示すると，図 3.1 のようになる．

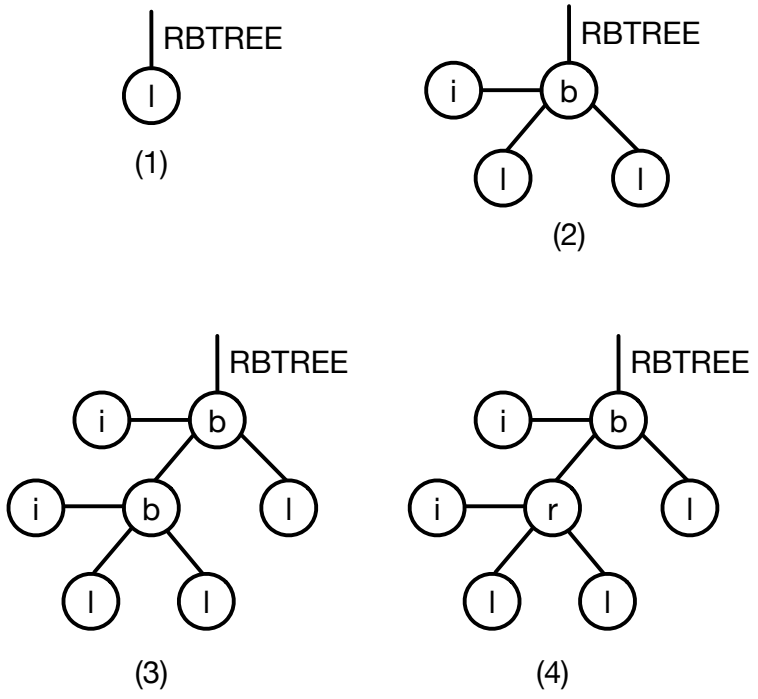

図 3.1　rbtree(RBTREE) 型のグラフ

このうち，(4) について，開始記号からこのグラフを得るための，生成規則の適用順の一例を，図 3.2 に示す．

3.3　グラフの型検査

グラフ X が，LMNtal Shape Type G 型であるかを検査する方法を示す．一般的なアルゴリズムを示す前に，具体例によって，型検査の流れを説明する．

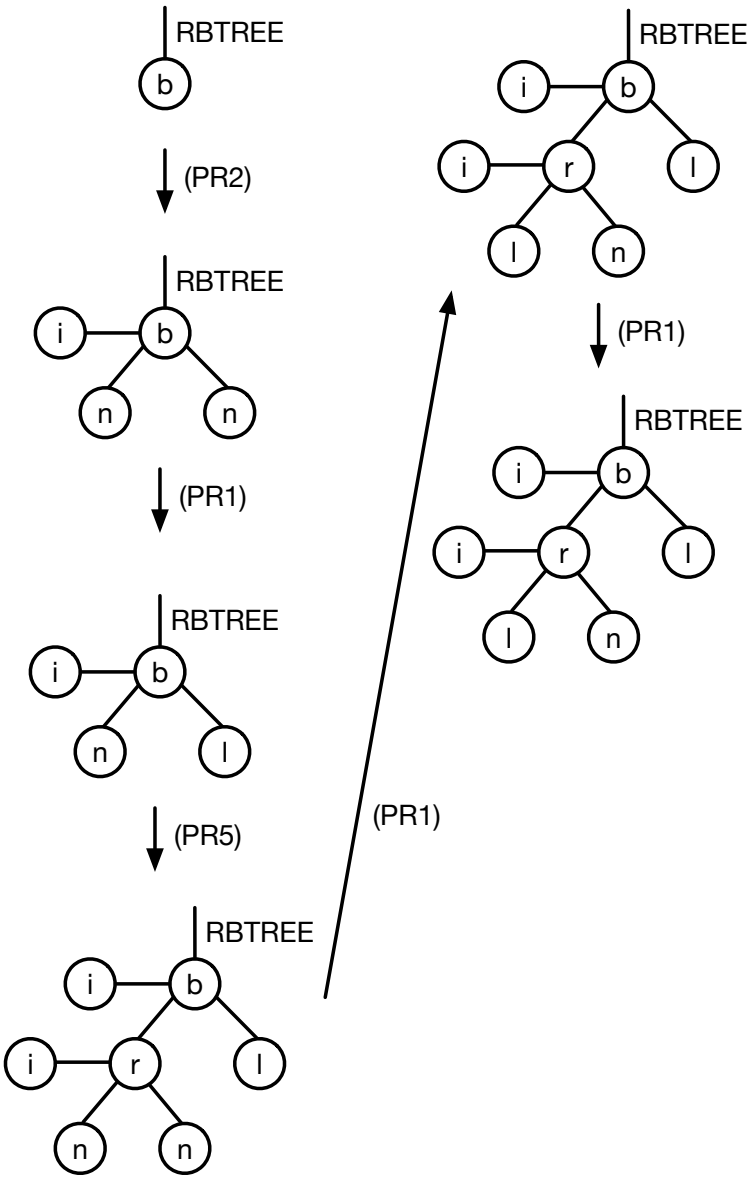

図 3.2　型検査における，生成規則の適用順の一つ

3.3.1　検査例

グラフ

$$\texttt{bnode}(\texttt{int}, \texttt{leaf}, \texttt{leaf}, \texttt{RBTREE})$$

が，rbtree(RBTREE) 型であることを検査する．このグラフを X とおくと，$X \in \mathrm{Graph}(\mathrm{Term}(\texttt{rbtree}))$ かつ $X \lhd \texttt{rbtree}$ であればよい．$\mathrm{Term}(\texttt{rbtree}) =$

$\{(\texttt{leaf}, 1), (\texttt{bnode}, 4), (\texttt{rnode}, 4), (\texttt{int}, 1), (\texttt{=}, 2)\}$ であるから，$X \in \mathrm{Graph}(\mathrm{Term}(G))$ である．

$X \lhd \texttt{rbtree}$ であることを確かめる．$X \lhd \texttt{rbtree}$ とは，X が，開始記号 $\texttt{bnode}(\texttt{RBTREE})$ から，生成規則の有限回の適用によって生成されることである．そこで，生成規則を逆向きに適用して，X が開始記号まで到達するかどうかを調べる．

例えば，X に，(PR3) を逆向きにしたルール

$$\texttt{leaf}(\texttt{PARENT}) \text{ :- } \texttt{bnode}(\texttt{PARENT})$$

を適用することで，グラフ

$$\texttt{bnode}(\texttt{int}, \texttt{bnode}, \texttt{leaf}, \texttt{RBTREE})$$

が得られる．また，このグラフにもう一度，(PR3) を逆向きにしたルールを適用することで，グラフ

$$\texttt{bnode}(\texttt{int}, \texttt{bnode}, \texttt{leaf}, \texttt{RBTREE})$$

が得られる．

このように，X に対する，生成規則の逆向き適用方法を列挙していくと，図 3.3 のようになる．

このうち，開始記号

$$\texttt{bnode}(\texttt{RBTREE})$$

に行き着く遷移，例えば

$$\begin{array}{c} \texttt{bnode}(\texttt{int}, \texttt{leaf}, \texttt{leaf}, \texttt{RBTREE}) \\ \downarrow \\ \texttt{bnode}(\texttt{int}, \texttt{leaf}, \texttt{node}, \texttt{RBTREE}) \\ \downarrow \\ \texttt{bnode}(\texttt{int}, \texttt{node}, \texttt{node}, \texttt{RBTREE}) \\ \downarrow \\ \texttt{bnode}(\texttt{RBTREE}) \end{array}$$

に着目する．この遷移を逆向きにたどることでただちに，開始記号 $\texttt{bnode}(\texttt{RBTREE})$ から，グラフ X を生成する方法を得ることができる．すなわち，$X \lhd \mathrm{rbtree}$ である．

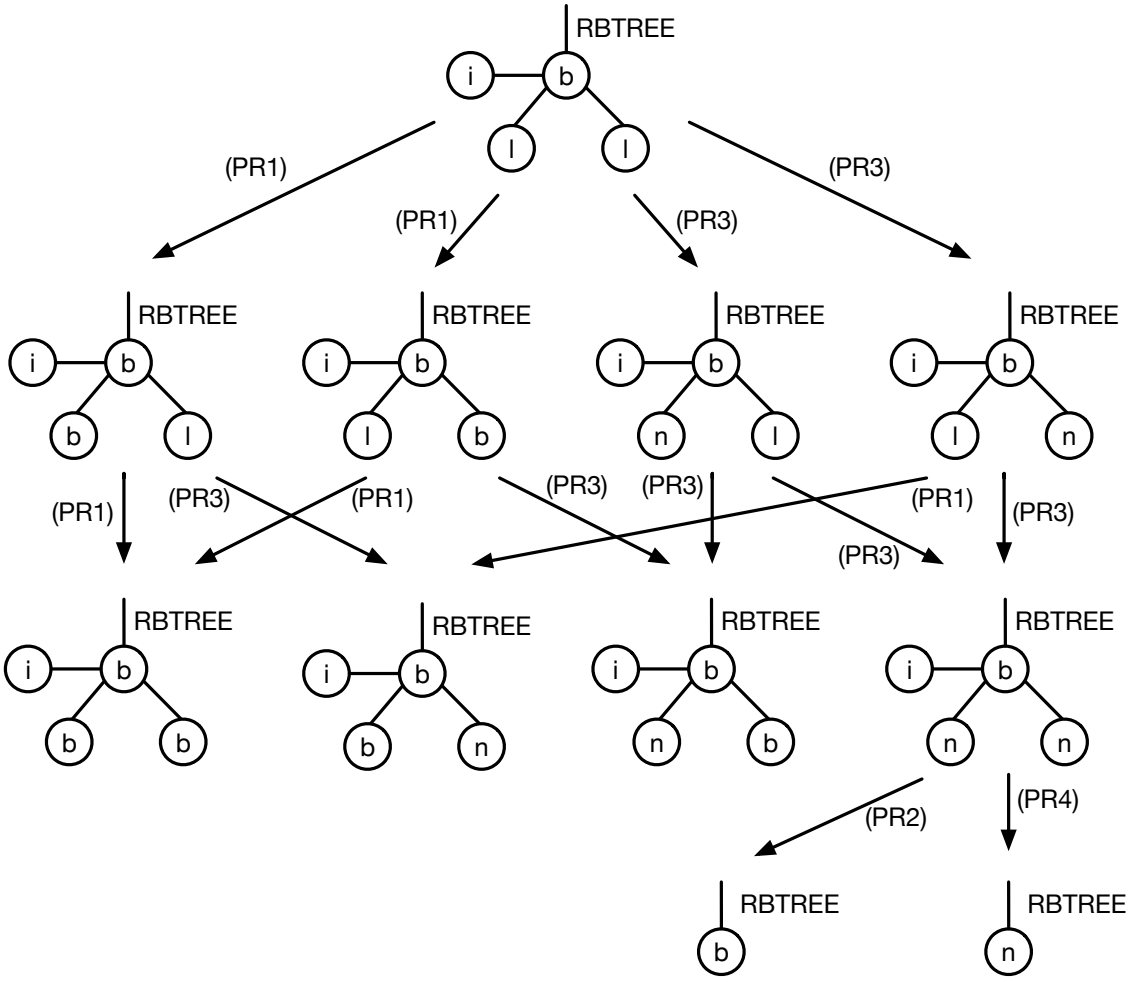

図 3.3　グラフの型検査における，生成規則の逆向き適用方法

3.3.2　アルゴリズム

では，グラフ X が G 型であるかどうか検査するアルゴリズムを示す．$X \in \mathrm{Graph}(\mathrm{Term}(G))$ であるかどうかは，X に出現する全てのファンクタが $\mathrm{Term}(G)$ に属するか調べればよい．その手続は自明であるので，記載を省略し，$X \lhd G$ であるかどうかを検査するアルゴリズムを示す．ただし，以下の手続き $\mathrm{Next}(X, R)$ が与えられているものとする．

Next(X:グラフ, R:ルールセット):

1: $\mathbf{X} \leftarrow X, R \rightarrow X', R$ なるすべてのグラフ X' の集合
2: $\mathcal{X} \leftarrow \mathbf{X}$ の，グラフの合同関係による商集合
3: **return** $\mathcal{X}$ の各々の同値類から，適当なグラフを選びだした集合

また，生成規則集合からルールセットへの写像 Reduction を，以下のように定義する．

$$\begin{aligned} \text{Reduction}(p(L_1, ..., L_n) \Rightarrow X) &= X \text{ :- } p(L_1, ..., L_n) \\ \text{Reduction}(Prod_1.Prod_2) &= \text{Reduction}(Prod_1), \text{Reduction}(Prod_2) \end{aligned}$$

では，$X \lhd G$ か検査する手続き HasType を示す．

HasType(X:グラフ, G:LMNtal Shape Type,
VisitedGraphs:グラフの有限集合):

1: **if** $\exists X' \in$ *VisitedGraphs* such that $X \equiv X'$ **then**
2: **return false**
3: **else if** $X \equiv s(LS_1, ..., LS_n)$ **then**
4: **return true**
5: **else**
6: **for all** $X' \in \text{Next}(X, \text{Reduction}(Prod))$ **do**
7: **if** HasType(X', G, *VisitedGraphs* $\cup \{X\}$) **then**
8: **return true**
9: **end if**
10: **end for**
11: **return false**
12: **end if**

HasType は，与えられたグラフに，与えられた文法の生成規則を逆向きに適用し，開始記号まで至るパスを深さ優先で探索するものである．

手続き HasType に，X, G, $\emptyset$ を与えることで，$X \lhd G$ であるか検査できる．

3.3.3 グラフの型検査の停止性

グラフの型検査のために，HasType(X, G, $\emptyset$) の呼び出しを行ったとき，この手続は停止することを示す.

まず，グラフから自然数への関数 NAtom を，以下のように定義する.

$$\begin{aligned} \mathrm{NAtom}(0) &= 0 \\ \mathrm{NAtom}(p(L_1,...,L_n)) &= \begin{cases} 0((p,n)=(=,2)\ \text{のとき}) \\ 1(\text{それ以外}) \end{cases} \\ \mathrm{NAtom}(X_1,X_2) &= \mathrm{NAtom}(X_1)+\mathrm{NAtom}(X_2) \end{aligned}$$

つまり，NAtom(X) は，2 価の＝アトムを除いた，X に含まれるアトム数である.

また，LMNtal Shape Type G に対し，G の生成規則の右辺または左辺に出現するファンクタ全体の集合を，Functors(G) で表す.

補題 1. 任意のグラフ X と X'，および任意の生成規則集合 *Prod* について，$X' \in \mathrm{Next}(X, \mathrm{Reduction}(Prod))$ ならば，$\mathrm{NAtom}(X') \leq \mathrm{NAtom}(X)$

証明：

生成規則の左辺は 1 つのアトムからなり，右辺は 1 つ以上のアトムからなっている．したがって，生成規則を逆適用して，2 価の =アトムを除くアトム数が増えることはないため成り立つ.

補題 2. 任意の LMNtal Shape Type G と，自然数 n と，リンクの有限集合 *Links* について，$\mathbf{X} = \{X \in \mathrm{Graph}(\mathrm{Functors}(G)) \mid \mathrm{NAtom}(X) = n$ かつ X は *Links* を自由リンクとして持つ $\}$ とおくと，$\mathbf{X}$ のグラフの合同関係による商集合は有限集合である

証明：

有限種類のファンクタだけを許して，有限個のアトムを用いてできる閉じたグラフは有限個である．自由リンクの出現を許すと，自由リンク名の異なるグラフは同値ではないから，可算無限個のグラフができるが，出現することのできる自由リンクを指定すれば，有限個のグラフしかできない.

補題 3. 任意のプロセス P と P' について，P :- P' ならば，P の持つ自由リンクの集合と，P' の持つ自由リンクの集合は等しい

証明：

(Equiv1)〜(Equiv8) に，自由リンクが消滅したり新たに出現したりする規則がないため成り立つ．

定理 1. 任意の G と，$X \in \mathrm{Graph}(\mathrm{Functors}(G))$ と，有限な *VisitedGraphs* について，$\mathrm{HasType}(X, G, \mathit{VisitedGraphs})$ は停止する

証明：

HasType が再帰呼び出しを行うとき，引数のグラフ X' は $\mathrm{Next}(X, \mathrm{Reduction}(\mathit{Prod}))$ の元である．したがって，補題 1 より，$\mathrm{NAtom}(X') \leq \mathrm{NAtom}(X)$ であるから，再帰の各段階で，引数グラフ X に対する $\mathrm{NAtom}(X)$ の値は，最初の呼び出しでの引数のグラフ X に対する $\mathrm{NAtom}(X)$ 以下に収まる．さらに，補題 3 と，Next の定義より，X' のもつ自由リンクの集合は，X のもつ自由リンクの集合から変化しない．したがって，再帰の各段階での，引数グラフ X のもつ自由リンクの集合は常に同じである．これらのことと，補題 2 から，HasType の再帰呼び出し各段階で，引数グラフとして与えられる可能性のあるグラフは，合同なものの違いを無視すれば有限個である．3 行目で，以前に与えられたグラフと合同なものが与えられると，探索を終了しているので，HasType の呼び出し回数は有限回に収まる．したがって，HasType は停止する．

系 1. グラフの型検査アルゴリズムの停止性：
任意の G と $X \in \mathrm{Graph}(\mathrm{Term}(G))$ について，$\mathrm{HasType}(X, G, \emptyset)$ は停止する

これは，定理 1 より直ちに成り立つ．

3.3.4 グラフの型検査の健全性

$\mathrm{HasType}(X, G, \emptyset)$ が true を返したならば，$X : G$ であることを示す．

補題 4. 任意のグラフ X と Y と，$\mathit{Rule}_X \equiv \mathit{Rule}_Y \equiv X_l$:- X_r なる任意のルール Rule_X と Rule_Y について，$X, \mathit{Rule}_X \to Y, \mathit{Rule}_Y$ ならば，$Y, (X_r$:- $X_l) \to X, (X_r$:- $X_l)$

証明：

$X, \mathit{Rule}_X \to Y, \mathit{Rule}_Y$ を仮定する．この時用いた遷移規則に関する帰納法で，$Y, (X_r$:- $X_l) \to X, (X_r$:- $X_l)$ を示す．

(Trans1) の場合：$X = X_1, X_2$，$Y = X_1', X_2$ とおく．帰納法の仮定より，$X_1', (X_r$:- $X_l) \to X_1, (X_r$:- $X_l)$ である．したがって，(Trans1) より，$Y, (X_r$:- $X_l) \to X, (X_r$:- $X_l)$ となる．

(Trans2) の場合：$X, Rule_X \equiv X', R_1$，$Y, Rule_Y \equiv Y', R_2$ とおく．ルールセットの合同変形で，ルールの個数は変化しないから，適当なルール $Rule_1$ と $Rule_2$ を用いて，$R_1 = Rule_1$，$R_2 = Rule_2$ とおける．プロセスの合同関係の定義より，$Rule_1 \equiv Rule_X \equiv X_l$:- X_r，$Rule_2 \equiv Rule_Y \equiv X_l$:- X_r である．したがって，帰納法の仮定より，$Y', (X_r$:- $X_l) \rightarrow X', (X_r$:- $X_l)$ であるから，(Trans2) より，$Y, (X_r$:- $X_l) \rightarrow X, (X_r$:- $X_l)$ となる．

(Trans3) の場合：$X_l = X, X_r = Y$ である．(Trans3) より，$Y, (Y$:- $X) \rightarrow X, (Y$:- $X)$ であるから，$Y, (X_r$:- $X_l) \rightarrow X, (X_r$:- $X_l)$ となる．

補題 5. 任意のグラフ X と，ルールセット R について，$X, R \rightarrow X', R$ ならば，あるルール C :- A とルールセット R' が存在し，$R \equiv (C$:- $A), R'$ かつ $X, (C$:- $A) \equiv X', (C$:- $A)$

証明：

$X, R \rightarrow X', R$ の証明図が与えられたとき，(Trans3) は必ず一回使用されている．(Trans3) に現れるルール，あるいはそれと合同なルールが，C :- A である．

補題 6. 任意の生成規則集合 *Prod* について，Reduction(*Prod*) $\equiv (X_l$:- $X_r), R$ であるならば，あるルールセット R' が存在して，$(X_r$:- $X_l), R' \equiv$ Induction(*Prod*)

証明：

Induction(*Prod*) は，Reduction(*Prod*) に含まれるルールを，全て逆向きにして分子を作ったルールセットと合同である．したがって，R に含まれるルールを，全て逆向きにして分子を作ったルールセットを R' と置けば，$(X_r$:- $X_l), R' \equiv$ Induction(*Prod*) となる．

定理 2. 任意の LMNtal Shape Type G と，$X \in$ Graph(Term(G)) なる X と，グラフの有限集合 *VisitedGraphs* について，HasType(X, G, *VisitedGraphs*) が true を返すなら，$X : G$ である

証明：

$X \in$ Graph(Term(G)) であるから，$X \lhd G$ が示せればよい．HasType が再帰呼び出しされた回数に関する帰納法で，これを示す．

HasType(X, G, *VisitedGraphs*) が，再帰呼び出しを行わずに true を返した場合

このとき，HasType(X, G, *VisitedGraphs*) は，4 行目で true を返している．$X \equiv s(LS1, ..., LSn)$ であるので，(Typing1) より，$X \lhd G$.

n 回の再帰呼び出しによって，HasType(X, G, *VisitedGraphs*) が true を返すならば，$X : G$ であると仮定する．HasType(X, G, *VisitedGraphs*) が $n+1$ 回の再帰呼び出しを

行って true を返した場合

再帰呼び出しが 1 回以上行われるので，HasType(X, G, *VisitedGraphs*) は，8 行目で true を返している．Next の定義より，$X, \mathrm{Reduction}(Prod) \rightarrow X', \mathrm{Reduction}(Prod)$ である．補題 5 より，あるルール X_l :- X_r とルールセット R が存在し，$\mathrm{Reduction}(Prod) \equiv (X_l$:- $X_r), R$ かつ $X, (X_l$:- $X_r) \rightarrow X', (X_l$:- $Xr)$．補題 4 より，$X', (X_r$:- $X_l) \rightarrow X, (X_r$:- $X_l)$．補題 6 より，$(X_r$:- $X_l), R' \equiv \mathrm{Induction}(Prod)$ なるルールセット R' をとる．(Trans1) より，$X', (X_r$:- $X_l), R' \rightarrow X, (X_r$:- $X_l), R'$．(Trans2) より，$X', \mathrm{Induction}(Prod) \rightarrow X, \mathrm{Induction}(Prod)$．HasType($X'$, G, *VisitedGraphs* $\cup \{X\}$) は，n 回の再帰呼び出しを行って true を返すので，帰納法の仮定より，$X' \lhd G$．したがって，(Typing2) より，$X \lhd G$.

系 2. HasType の健全性：
任意の LMNtal Shape Type G と，$X \in \mathrm{Graph}(\mathrm{Term}(G))$ なる X について，HasType(X, G, $\emptyset$) が true を返すなら，$X : G$ である

これは，定理 2 より直ちに成り立つ.

3.4 ルール型検査

文脈自由な型、すなわち、全ての生成規則の左辺がひとつのアトムだけからなるような型については、与えられたルールの型保存性が検査できる。検査は以下の 2 ステップで構成される。

1. 検査対象ルール左辺を含む t 型のグラフを生成する、証明図のパターンを全て列挙する
2. 全ての証明図パターンに対し、検査対象ルールの左辺グラフを右辺グラフに置換したグラフが t 型であることの証明が、パターンを元に構成できるか調べる

ルール型検査の例として，以下のスキップリスト [4] 型を保存するルールを考える.

```
skiplist(List1,List2).
skiplist(L1,L2)  ⇒  nil(L1,L2)                    (Prod0)
     nil(L1,L2)  ⇒  node1(X,L1),nil(X,L2)         (Prod1)
     nil(L1,L2)  ⇒  node2(X,Y,L1,L2),nil(X,Y)     (Prod2)
```

以降では、リンクの集合を $\boldsymbol{L}$ というように太字で表記する。

3.4.1 ステップ 1：証明図パターンの列挙

ステップ 1 を実現するには、検査対象ルール左辺に、生成規則を逆向きにしたルールを適用し、開始記号まで遷移させればよい。ただし、検査対象ルール左辺以外のグラフが未知であるので、逆向きルール中の一部（ひとつ以上）のアトムと反応することを許す。

つまり、ルール左辺 $LHS[\boldsymbol{L_1}]$ に対し、証明図の最下部

$$G_n[\boldsymbol{L_n} \cup \boldsymbol{L}], LHS[\boldsymbol{L_n}] \lhd t(\boldsymbol{L})$$

からスタートし、

$$G_1[\boldsymbol{L_1} \cup \boldsymbol{L}], t(\boldsymbol{L_1}) \lhd t(\boldsymbol{L})$$

に到達するまで、以下を繰り返す。

1. 外部グラフ $G_i[\boldsymbol{L_i} \cup \boldsymbol{L}]$ のみを、生成規則によって 0 回以上還元する
2. 検査対象ルールの左辺グラフの一部である $LHS_i[\boldsymbol{L_i}]$ の部分グラフ $LHS'_i[\boldsymbol{L'_i}]$（空であってはならない）と、外部グラフの部分グラフ $G'_{i-1}[\boldsymbol{L'_{i-1}}]$（空でもよい）からなる分子を、生成規則によって還元する。ここで、$LHS_i[\boldsymbol{L_i}]$ の残りを $LHS''_i[\boldsymbol{L''_i}]$、還元によってできるグラフ（これは一つのアトムのみから生る）を $G''_{i-1}[\boldsymbol{L'_i} \triangle \boldsymbol{L'_{i-1}}]$ とする。

これらの操作によって、部分的な証明図

$$\dfrac{\dfrac{\dfrac{G_{i-1}[\boldsymbol{L_i} \cup \boldsymbol{L} \setminus \boldsymbol{L'_{i-1}}], G''_{i-1}[\boldsymbol{L'_i} \triangle \boldsymbol{L'_{i-1}}], LHS''_i[\boldsymbol{L''_i}] \lhd t(\boldsymbol{L})}{G_{i-1}[\boldsymbol{L_i} \cup \boldsymbol{L} \setminus \boldsymbol{L'_{i-1}}], G'_{i-1}[\boldsymbol{L'_{i-1}}], LHS_i[\boldsymbol{L_i}] \lhd t(\boldsymbol{L})}}{\boxed{\text{Subproof}}}}{G_i[\boldsymbol{L_i} \cup \boldsymbol{L}], LHS_i[\boldsymbol{L_i}] \lhd t(\boldsymbol{L})}$$

を得る。なお、2 の還元において、複数通りの還元が可能であったり、還元が不可能である場合がある。また、還元の結果得られた明示的なグラフが、すでに得られた明示的なグラフと同型である場合、その還元は考えない。

これを繰り返して証明図を構成した後、最後に

$$\text{skiplist(List1, List2)} \lhd \text{skiplist(List1, List2)}$$

$$\boxed{\text{Subproof 0}}$$

$$\dfrac{G_1[\text{List1, List2, X, Y}], \text{skiplist(X, Y)} \lhd \text{skiplist(List1, List2)}}{G_1[\text{List1, List2, X, Y}], \text{nil(X, Y)} \lhd \text{skiplist(List1, List2)}}\ (\text{Prod}_0)$$

$$\boxed{\text{Subproof 1}}$$

$$\dfrac{G_2[\text{List1, List2, X, Y}], \text{nil(X, Y)} \lhd \text{skiplist(List1, List2)}}{G_2[\text{List1, List2, X, Y}], \text{nil(X1, Y1), node2(X1, Y1, X, Y)} \lhd \text{skiplist(List1, List2)}}\ (\text{Prod}_2)$$

$$\boxed{\text{Subproof 2}}$$

$$\dfrac{G_3[\text{List1, List2, X, Y}], \text{nil(X1, Y1), node2(X1, Y1, X, Y)} \lhd \text{skiplist(List1, List2)}}{\begin{array}{c}G_3[\text{List1, List2, X, Y}], \text{nil(X3, Y2), node2(X1, Y1, X, Y), node2(X3, Y2, X1, Y1)} \\ \lhd \text{skiplist(List1, List2)}\end{array}}\ (\text{Prod}_2)$$

$$\boxed{\text{Subproof 3}}$$

$$\begin{array}{c}G_4[\text{List1, List2, X, Y, X3, Y2}], \text{node2(X1, Y1, X, Y), node2(X3, Y2, X1, Y1)} \\ \lhd \text{skiplist(List1, List2)}\end{array}$$

図 3.4　ルールの左辺グラフの生成

$$\dfrac{\dfrac{t(\boldsymbol{L'}) \lhd t(\boldsymbol{L})}{\boxed{\text{Subproof}}}}{G_1[\boldsymbol{L_1} \cup \boldsymbol{L}], t(\boldsymbol{L_1}) \lhd t(\boldsymbol{L})}$$

を加えて、ルール左辺を含むグラフの生成に関する証明図を得ることができる。

例えば、レベル 2 のスキップリストについて，型を保存するルール

```
node2(X1,Y1,X,Y), node2(X2,Y2,X1,Y1)
:- node1(X1,X), node2(X2,Y,X1,Y2)
```

の場合、ステップ 1 によって、図 3.4 に示す証明図のみが得られる。検査対象ルールの左辺グラフ node1(X1,X), node2(X2,Y,X1,Y2) を含むスキップリストは、全てこのパターンで生成することができる。

3.4.2　ステップ 2：証明図パターンの変形

ステップ 1 で得られた証明図の、一番下に現れる外部グラフ $G_n[\boldsymbol{L_n} \cup \boldsymbol{L}]$ は、そのパターンの証明図で生成される全ての外部グラフを代表している。従って、すべての証明図

パターンの各々に対して、それを元に、

$$G_n[\boldsymbol{L_n} \cup \boldsymbol{L}], RHS[\boldsymbol{L_n}] \lhd t(\boldsymbol{L})$$

の証明図を構成できれば、任意の $t(\boldsymbol{L})$ 型のグラフについて、ルール LHS :- RHS は型を保存する事が分かる。

ここで、このことは型保存のための十分条件であるが、必要条件ではないことに注意する。たとえば、二つの証明図パターンが得られ、証明図最下部に現れる外部グラフが、それぞれ G, G' だったとする。もし、 G として考えられるグラフの集合と、 G' として考えられるグラフの集合が一致するなら、二つのうち一つの証明図パターンに対して、書換え後のグラフが t 型である証明が得られれば十分である。必ずしも、両方に対して証明が得られる必要はない。

では、左辺グラフを含むグラフの証明図パターンを元に、書き換え後のグラフの証明図を構成する方法を示す。まず、$i \geq 1$ なる各々の i に対し、

$$\dfrac{\dfrac{G_i[\boldsymbol{L_{i+1}} \cup \boldsymbol{L} \setminus \boldsymbol{L'_i}], G'_i[\boldsymbol{L'_i}], LHS_{i+1}[\boldsymbol{L_{i+1}}] \lhd t(\boldsymbol{L})}{\boxed{\text{Subproof}_n}}}{G_{i+1}[\boldsymbol{L_{i+1}} + \boldsymbol{L}], LHS_{i+1}[\boldsymbol{L_{i+1}}] \lhd t(\boldsymbol{L})}$$

の形の部分的な証明図が得られているとする。

右辺グラフ $RHS_n[\boldsymbol{L_n}]$ と、グラフ $G'_1[\boldsymbol{L'_1}], ..., G'_{n-1}[\boldsymbol{L'_{n-1}}]$ からなる分子を考える。ただし、自由リンクは適切に接続されているものとする。 この分子をグラフ型検査にかける。ただし、型の持つ自由リンクの接続方法は問わない。

型検査に成功し、グラフがアトム t まで還元されると、グラフの自由リンクと型の自由リンクの対応関係がわかるので、それに従って、証明図中のグラフの自由リンク名を書き換える。

型検査に失敗した場合は、ルール型検査は失敗したとみなす。

スキップリストの例だと、右辺グラフ node1(X1, X), node2(X2, Y, X1, Y2) に付加すべきグラフは、 nil(X2, Y2) である。これらからなる分子を型検査にかけると、図 3.5 に示す証明図を得る。

ここでできた右辺グラフの証明図と、左辺グラフの証明図の Subproof を適切に組み合わせることで、$G_n[\boldsymbol{L_n} \cup \boldsymbol{L}], RHS[\boldsymbol{L_n}] \lhd t(\boldsymbol{L})$ の証明図を構成できる。

スキップリストの例だと、図 3.6 に示す証明図を得ることができる。

したがって、グラフ $G'_1[\boldsymbol{L'_1}], ..., G'_{n-1}[\boldsymbol{L'_{n-1}}]$ を、型の持つ自由リンクの接続方法を問わず型検査して、検査が成功すれば、ルールは型を保存することが分かる。

$$\dfrac{\dfrac{\dfrac{\text{skiplist(List1, List2)} \lhd \text{skiplist(List1, List2)}}{\text{nil(List1, List2)} \lhd \text{skiplist(List1, List2)}}\,(\text{Prod}_0)}{\text{node1(X1, List1), nil(X1, List2)} \lhd \text{skiplist(List1, List2)}}\,(\text{Prod}_1)}{\text{node1(X1,List1), node2(X2, List2, X1, Y2), nil(X2, Y2)} \lhd \text{skiplist(List1, List2)}}\,(\text{Prod}_2)$$

図 3.5　スキップリストのルールの右辺グラフ型検査

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\text{skiplist(List1, List2)} \lhd \text{skiplist(List1, List2)}}{\boxed{\text{Subproof 0}}}}{G_1\text{[List1, List2, X, Y], skiplist(X, Y)} \lhd \text{skiplist(List1, List2)}}}{G_1\text{[List1, List2, X, Y], nil(X, Y)} \lhd \text{skiplist(List1, List2)}}\,(\text{Prod}_0)}{\boxed{\text{Subproof 1}}}}{G_2\text{[List1, List2, X, Y], nil(X, Y)} \lhd \text{skiplist(List1, List2)}}}{G_2\text{[List1, List2, X, Y], node1(X1, X), nil(X1, Y)} \lhd \text{skiplist(List1, List2)}}\,(\text{Prod}_1)}{\boxed{\text{Subproof 2}}}}{G_2\text{[List1, List2, X, Y], node1(X1, X) nil(X1, Y)} \lhd \text{skiplist(List1, List2)}}}{G_2\text{[List1, List2, X, Y], node1(X1, X), node2(X2, Y2, X, Y), nil(X2, Y2)} \lhd \text{skiplist(List1, List2)}}\,(\text{Prod}_2)}{\boxed{\text{Subproof 3}}}}{G_3\text{[List1, List2, X, Y, X2, Y2], node1(X1, X), node2(X2, Y2, X1, Y)} \lhd \text{skiplist(List1, List2)}}$$

図 3.6　ルールの右辺グラフの生成

第 4 章

まとめと今後の課題

4.1 まとめ

本研究では，LMNtal へ Shape Type を導入することにより，グラフの型検査や，ルールの型保存制検査が可能となることを示した.

4.2 今後の展開

4.2.1 ルール型検査の完全性について

グラフの型検査アルゴリズムの完全性に関しては考察していない．[2] によれば，ルールの型検査は，文脈自由言語の包含関係を判定する問題に帰着するため，完全ではないとされている．事実，左辺グラフについて複数の証明図パターンが得られる場合，完全性が失われる．しかし，型検査が偽陰性を示す実用的なルールは見つかっていない.

4.2.2 型体系の拡張

LMNtal Shape Type の拡張として，文脈依存文法への拡張が考えられる．つまり，生成規則の左辺に，2 つ以上のアトムからなるグラフの出現を許す，ということである．これを行えば，例えば，赤黒木の持つ，「任意のノードについて，そのノードから葉までの任意の路に含まれる黒のノードの数は一定である」という性質を型で表現することができる.

文脈依存文法へ拡張すると，グラフの型検査の停止性が失われる．これは，停止性が，「生成規則の逆向き適用により，アトム数が増えることはない」（補題 1）という事実に依

存しているためである．停止性を保つためには，許される生成規則の形を適切に制限する必要がある．また，ルール型検査において，証明図パターン生成で用いる部分マッチの性質上，型検査が停止しないケースがあることがわかっている．

他には，多相型への拡張が考えられる．データ構造のもつデータとして，「int 型の任意の値」などを指定できれば便利である．さらに，生成規則に数値制約を導入することで，「二分探索木である」といった性質も型で表現できる．関数型言語においては，数値制約を持つ型（一種の依存型）として，Liquid Type [5] といった型が考案されている．

さらには，ルールを複数回適用しても，LMNtal Shape Type が保存されることを検査したい．LMNtal で実際にデータ構造を操作するときは，1 回のルール適用で操作が完了することは稀である．したがって，ルール適用の途中で型が崩れても，操作が完了した後には型が保存されるという性質を保証できることは重要である．これは，操作途中の型をユーザが指定，あるいは推論し，操作の最初と最後で型が変化することを検査できればよい．

謝辞

研究を進めるにあたって、ご指導ご鞭撻頂いた上田和紀教授、貴重な意見をくれた後輩たちに感謝します。また、不甲斐ない私を最後まで支えてくれた両親に感謝します。

2017 年 1 月 25 日　吉元 佑介

参考文献

[1] 上田和紀, 加藤紀夫, 言語モデル LMNtal, コンピュータソフトウェア, Vol. 21, No. 2, pp. 126–142, 2004.

[2] Pascal Fradet, Daniel Le Metayer, Shape Types, in *POPL'97: Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on principles of Programming Languages*, pp. 27–39, 1997.

[3] R. Bayer, Symmetric Binary B-Trees: Data Structure and Maintenance Algorithms, *Acta Informatica*, Vol. 1, Issue 4, pp. 290–306, 1972.

[4] William Pugh, Skip Lists: A Probabilistic Alternative to Balanced Trees, *Communications of the ACM*, Vol. 33, No. 6, pp. 668–676, 1990.

[5] Patrick, R., Kawaguchi, M. and Jhara, R., Liquid Types, in *PLDI: Proceedings of the Symposium on Programming Language Design and Implementation*, pp. 158–169, 2008.

外部発表

[1] 吉元佑介, 上田和紀, グラフ書換え系における静的グラフ型検査, 日本ソフトウェア科学会第 32 回大会, PPL3-3, 東京, 2015 年 9 月 9 日 (8 pages).